

Fast Constructions of Light-Weight Spanners for General Graphs

Michael Elkin *

Shay Solomon *[†]

Abstract

Since the pioneering works of Peleg and Schäffer [29], Althöfer et al. [4], and Chandra et al. [11], it is known that for every *weighted* undirected n -vertex m -edge graph $G = (V, E)$, and every integer $k \geq 1$, there exists a $(2k - 1)$ -spanner with $O(n^{1+1/k})$ edges and weight $O(k \cdot n^{(1+\epsilon)/k}) \cdot \omega(MST(G))$, for any $\epsilon > 0$. (Here $\omega(MST(G))$ stands for the weight of the minimum spanning tree of G .) Nearly linear time algorithms for constructing $(2k - 1)$ -spanners with nearly $O(n^{1+1/k})$ edges were devised in [9, 34, 33]. However, these algorithms fail to guarantee any meaningful upper bound on the weight of the constructed spanners.

To our knowledge, there are only two known algorithms for constructing sparse and light spanners for general graphs. One of them is the greedy algorithm of Althöfer et al. [4], analyzed by Chandra et al. [11]. The drawback of the greedy algorithm is that it requires $O(m \cdot (n^{1+1/k} + n \cdot \log n))$ time. The other algorithm is due to Awerbuch et al. [6], from 1991. It constructs $O(k)$ -spanners with $O(k \cdot n^{1+1/k} \cdot \Lambda)$ edges, weight $O(k^2 \cdot n^{1/k} \cdot \Lambda) \cdot \omega(MST(G))$, within time $O(m \cdot k \cdot n^{1/k} \cdot \Lambda)$, where Λ is the logarithm of the aspect ratio of the graph.

The running time of both these algorithms is unsatisfactory. Moreover, the usually faster algorithm of [6] pays for the speedup by significantly increasing both the stretch, the sparsity, and the weight of the resulting spanner.

In this paper we devise an efficient algorithm for constructing sparse and light spanners. Specifically, our algorithm constructs $((2k - 1) \cdot (1 + \epsilon))$ -spanners with $O(k \cdot n^{1+1/k})$ edges and weight $O(k \cdot n^{1/k}) \cdot \omega(MST(G))$, where $\epsilon > 0$ is an arbitrarily small constant. The running time of our algorithm is $O(k \cdot m + \min\{n \cdot \log n, m \cdot \alpha(n)\})$. Moreover, by slightly increasing the running time we can reduce the other parameters. These results address an open problem from the ESA'04 paper by Roditty and Zwick [34].

*Department of Computer Science, Ben-Gurion University of the Negev, POB 653, Beer-Sheva 84105, Israel.

E-mail: {elkinm, shayso}@cs.bgu.ac.il

Both authors are supported by the BSF grant No. 2008430 and by the ISF grant No. 87209011. In addition, both authors are partially supported by the Lynn and William Frankel Center for Computer Sciences.

[†]This research has been supported by the Clore Fellowship grant No. 81265410.

1 Introduction

1.1 Centralized Algorithms. Given an undirected weighted graph $G = (V, E)$ and a parameter $t \geq 1$, a subgraph $H = (V, E')$ of G ($E' \subseteq E$) is called a t -*spanner* if for every edge $e = (u, v) \in E$, $\text{dist}_H(u, v) \leq t \cdot \text{dist}_G(u, v)$. (Here $\text{dist}_G(u, v)$ stands for the distance between u and v in the graph G .) Graph spanners were introduced in 1989 by Peleg and Schäffer [29] and Peleg and Ullman [30], who showed that for every unweighted n -vertex graph $G = (V, E)$ and an integer parameter $k \geq 1$, there exists an $O(k)$ -spanner with $O(n^{1+1/k})$ edges. Althöfer et al. [4] improved and generalized these results. They analyzed the natural greedy algorithm for constructing graph spanners, and showed that for every n -vertex *weighted* graph $G = (V, E)$ and an integer parameter $k \geq 1$, this algorithm constructs a $(2k - 1)$ -spanner with $O(n^{1+1/k})$ edges. (This is near-optimal [29].) They also showed that the weight of the resulting spanner is $O(n/k) \cdot \omega(MST(G))$. (We will use the normalized notion of weight, called *lightness*, which is the ratio between the weight of the spanner and $\omega(MST(G))$.) In SoCG'92, Chandra et al. [11] improved the lightness bound, and showed that spanners obtained by the greedy algorithm have lightness $O(k \cdot n^{(1+\epsilon)/k})$, for an arbitrarily small $\epsilon > 0$. However, the running time of their algorithm is $O(m \cdot (n^{1+1/k} + n \cdot \log n))$, where m stands for $|E|$. Around the same time Awerbuch et al. [6] devised an algorithm that constructs $O(k)$ -spanners with $O(k \cdot n^{1+1/k} \cdot \Lambda)$ edges and lightness $O(k^2 \cdot n^{1/k} \cdot \Lambda)$, where Λ is the logarithm of the aspect ratio of the input graph. The running time of the algorithm of [6] is $O(m \cdot k \cdot n^{1/k} \cdot \Lambda)$.

In the two decades that passed since the results of [29, 30, 4, 11, 6] graph spanners turned out to be extremely useful. Among their applications is compact routing [30, 31, 36], distance oracles and labels [28, 35, 33], network synchronization [5], and computing almost shortest paths [13, 34, 16, 19, 20]. Graph spanners also became a subject of intensive research for their own sake [13, 18, 16, 9, 37, 20, 19, 38, 32, 8].

In particular, a lot of research attention was devoted to devising efficient algorithms for constructing sparse spanners for weighted graphs. Cohen [13] devised a randomized algorithm for constructing $((2k - 1) \cdot (1 + \epsilon))$ -spanners with $O(k \cdot n^{1+1/k})$ edges. Her algorithm requires expected $O(m \cdot n^{1/k} \cdot k)$ time. Improving upon [13], Baswana and Sen [9] devised a randomized algorithm that constructs $(2k - 1)$ -spanners with expected $O(k \cdot n^{1+1/k})$ edges, within expected $O(k \cdot m)$ time. Roditty et al. [33] derandomized this algorithm without any loss in parameters, or in running time. Roditty and Zwick [34] devised a deterministic algorithm for constructing $(2k - 1)$ -spanners with $O(n^{1+1/k})$ edges in $O(k \cdot n^{2+1/k})$ time. In the discussion section at the end of their paper Roditty and Zwick [34] write:

“Another interesting property of the (original) greedy algorithm, shown by [11], is that the total weight of the edges in the $(2k - 1)$ -spanner that it constructs is at most $O(n^{(1+\epsilon)/k} \cdot \omega(MST(G)))$,¹ for any $\epsilon > 0$. Unfortunately, this property no longer holds for the modified greedy algorithm. Again, it is an interesting open problem to obtain an efficient spanner construction algorithm that does have this property.”

In the current paper we devise such a construction. Specifically, our algorithm constructs $((2k - 1) \cdot (1 + \epsilon))$ -spanners with $O(k \cdot n^{1+1/k})$ edges and lightness $O(k \cdot n^{1/k})$, and does so in time $O(k \cdot m + \min\{n \cdot \log n, m \cdot \alpha(n)\})$, where $\alpha(n)$ is an inverse Ackermann function. In other words, the running time of our algorithm is near-optimal and is drastically better than the running time $O(m \cdot (n^{1+1/k} + n \cdot \log n))$ of [11] and than that of [6] ($O(m \cdot k \cdot n^{1/k} \cdot \Lambda)$). We pay for this speed up by a small increase (by a factor of $(1 + \epsilon)$) in the stretch and a small increase (by a factor of k) in the number of edges. (This comparison is with [11]. Our algorithm strictly outperforms the algorithm of [6].) We also have another variant of our algorithm with a slightly higher running time ($O(k \cdot n^{2+1/k})$), and with $O(n^{1+1/k})$ edges and lightness $O(k \cdot n^{1/k})$.

Note that the relationship between the stretch and lightness in both our results is essentially the same as in the state-of-the-art bound [11]. Specifically, in our result the slack factor $(1 + \epsilon)$ appears in the stretch, while in [11] it appears in the exponent of the lightness. The number of edges in our slower construction (that runs in $O(k \cdot n^{2+1/k})$ time) is the same as in [11]. The faster variant of our algorithm (that runs in near-optimal time of $O(k \cdot m + \min\{n \cdot \log n, m \cdot \alpha(n)\})$), pays for the speedup by increasing

¹Actually, the weight in [11] is $O(k \cdot n^{(1+\epsilon)/k} \cdot \omega(MST(G)))$.

results	stretch	number of edges	lightness	running time
Althöfer et al. [4]	$2k - 1$	$O(n^{1+1/k})$	$O(n/k)$	$O(m \cdot n^{1+1/k})$
Chandra et al. [11]	$2k - 1$	$O(n^{1+1/k})$	$O(k \cdot n^{(1+\epsilon)/k})$	$O(m \cdot (n^{1+1/k} + n \cdot \log n))$
Awerbuch et al. [6]	$O(k)$	$O(k \cdot n^{1+1/k} \cdot \Lambda)$	$O(k^2 \cdot n^{1/k} \cdot \Lambda)$	$O(m \cdot k \cdot n^{1/k} \cdot \Lambda)$
Our faster construction	$(2k - 1) \cdot (1 + \epsilon)$	$O(k \cdot n^{1+1/k})$	$O(k \cdot n^{1/k})$	$O(k \cdot m + \min\{n \cdot \log n, m \cdot \alpha(n)\})$
Our slower construction	$(2k - 1) \cdot (1 + \epsilon)$	$O(n^{1+1/k})$	$O(k \cdot n^{1/k})$	$O(k \cdot n^{2+1/k})$

Table 1: A concise comparison of previous and our constructions of light spanners. All the constructions mentioned in this table are deterministic. Our results are indicated by bold fonts.

the number of edges by a factor of k . See Table 1 for a concise comparison of our and previous results on light spanners. (The lightness of all other spanner constructions [13, 9, 33, 34] is unbounded.)

Chandra et al. [11] also showed that the greedy algorithm gives rise to a construction of $O(\log^2 n)$ -spanners with $O(n)$ edges and constant lightness, and to a construction of $O(\log n)$ -spanners with $O(n)$ edges and lightness $O(\log n)$. The running time of these constructions is $O(m \cdot n \cdot \log n)$. Our algorithm also constructs spanners with the same (up to constant factors) parameters. The running time required by our algorithm to construct these spanners is $O(n^2 \cdot \log n)$.

1.2 Streaming Algorithms. In the streaming model of computation the input graph $G = (V, E)$ arrives as a “stream”, i.e., the algorithm reads edges one after another. The algorithm is required to process edges efficiently, and to store only a limited amount of information. In the context of computing spanners the natural memory limitation is the size of the spanner. Multi-pass streaming algorithms also allow several (ideally, just a few) passes over the input stream.

The streaming model of computation was introduced by Alon et al. [2] and by Feigenbaum et al. [21]. The study of graph problems in the streaming model was introduced by Feigenbaum et al. [20]. In particular, Feigenbaum et al. [20] devised a randomized one-pass streaming algorithm for computing a $(2k + 1)$ -spanner with expected $O(k \cdot \log n \cdot n^{1+1/k})$ edges, using $O(k \cdot \log n \cdot n^{1/k})$ processing time-per-edge. This result was improved in [17], who devised a randomized one-pass streaming algorithm that computes $(2k - 1)$ -spanners with expected $O(k \cdot n^{1+1/k})$ edges, using $O(1)$ processing time-per-edge. See also [7]. Elkin and Zhang [19] devised a multi-pass streaming algorithm for constructing sparse $(1 + \epsilon, \beta)$ -spanners. The number of passes in their algorithm is $O(\beta)$.

To our knowledge, there are currently no efficient streaming algorithms for computing *light* spanners. We show that our algorithm can be implemented efficiently in the streaming model *augmented with the sorting primitives* (henceforth, *augmented streaming model*). This model, introduced by Aggarwal et al. [1] in FOCS’94, allows to have “sort passes”. As a result of a sort pass, in consequent passes one can assume that the input stream that the algorithm reads is sorted. (See [1] for the justification of this model. The authors in [1] argue that “streaming computations with an added sorting primitive are a natural and efficiently implementable class of massive data set computations”.)

The algorithm of Chandra et al. [11] can be viewed as an algorithm in this model. After the initial sorting pass, it requires one pass over the input stream. As a result it constructs a $(2k - 1)$ -spanner with $O(n^{1+1/k})$ edges and lightness $O(k \cdot n^{(1+\epsilon)/k})$, for an arbitrarily small $\epsilon > 0$. The processing time-per-edge of this algorithm is, however, $O(n^{1+1/k})$, i.e., prohibitively large.

We show that a variant of our algorithm computes $((2k - 1) \cdot (1 + \epsilon))$ -spanners with expected $O(k \cdot n^{1+1/k})$ edges and expected lightness $O(k^2 \cdot n^{1/k})$. It performs two passes over the input stream, that follow an initial sorting pass. In the first pass the worst-case (resp., amortized) processing time-per-edge of our algorithm is $O(\frac{\log n}{\log \log n})$ (resp., $O(\alpha(n))$). The processing time-per-edge of our algorithm in its second pass over the input stream is $O(1)$.

1.3 Our Techniques. Our algorithm is based on a transformation, which given a black-box construction of sparse (possibly heavy) spanners with a certain stretch t , efficiently produces sparse and *light* spanners with roughly the same stretch. We use this transformation in conjunction with a number of known

algorithms that produce sparse spanners, but do not provide any bound on their lightness.

Our transformation generalizes a *metric transformation* from [11]. Specifically, the metric transformation of [11] converts constructions of sparse spanners for metrics into constructions of sparse and light spanners (for the same metric). The *generalized transformation* that we devise applies to weighted not necessarily complete graphs. (Observe that a metric can be viewed as a complete weighted graph.)

There are a number of technical difficulties that we overcome in our way to the generalized transformation. Next, we briefly discuss one of them. The construction of [11] hierarchically partitions the point set of the input metric into clusters. Then it selects a representative point from each cluster, and invokes its input black-box construction of sparse spanners on the metric induced by the representatives. One can try to mimic this approach in graphs by replacing each missing metric edge between representatives by the shortest path between them. This approach, however, is doomed to failure, as the overall number of edges taken into the spanner in this way might be too large. To overcome this difficulty we carefully select *representative edges* which are inserted into a certain auxiliary graph. Then the black-box input construction is applied to the auxiliary graph. As a result we obtain a spanner of the auxiliary graph, which we call *auxiliary spanner*. This auxiliary spanner $\mathcal{Q} = (\mathcal{U}, \mathcal{E})$ is a graph over a new vertex set \mathcal{U} , i.e., \mathcal{U} is not a subset of the original vertex set V . Next, we “project” the auxiliary spanner \mathcal{Q} onto the original graph, i.e., we translate edges of \mathcal{E} into edges of the original edge set E . This needs to be done carefully, to avoid blowing up the stretch and lightness. Also, it is crucial that this translation procedure will be efficient. Interestingly, we do not project vertices of \mathcal{U} onto vertices of V , but rather edges of \mathcal{E} onto edges of E . In particular, for a vertex $u \in \mathcal{U}$ and two edges $(u, x), (u, y) \in \mathcal{E}$, they may be translated into two vertex-disjoint edges $(u', x'), (u'', y') \in E$. As a result a path in \mathcal{Q} does not translate into a path in G , but rather into a collection of possibly vertex-disjoint edges. We show that these edges can be carefully glued into a path. This gluing, however, comes at a price of slightly increasing the stretch.

1.4 Related Work. The large body of work on constructing graph spanners efficiently was already discussed in Section 1.1. The problem of constructing light spanners efficiently was also studied in the context of *geometric spanners*. See [15] and [23], and the references therein.

1.5 Organization. In Section 2 we present and analyze our algorithm in the centralized model of computation. The algorithm is described in Section 2.1, and its analysis appears in Section 2.2. In Section 3 we present a few variants of our basic algorithm (from Section 2.1). In particular, the streaming variant of our algorithm is presented in Section 3.5.

1.6 Preliminaries. We will use the following results as a black-box (we write $n = |V|, m = |E|$).

Theorem 1.1 [24] [unweighted graphs] *For any unweighted graph $G = (V, E)$ and any integer $k \geq 1$, a $(2k - 1)$ -spanner with $O(n^{1+1/k})$ edges can be built in $O(m)$ time.*

Theorem 1.2 [9, 33] [weighted graphs I] *For any weighted graph $G = (V, E)$ and any integer $k \geq 1$, a $(2k - 1)$ -spanner with $O(k \cdot n^{1+1/k})$ edges can be built in $O(k \cdot m)$ time.*

Remark: The algorithm of [9] is randomized, but was later derandomized in [33]. Henceforth, the algorithm provided by Theorem 1.2 is deterministic.

Theorem 1.3 [34] [weighted graphs II] *For any weighted graph $G = (V, E)$ and any integer $k \geq 1$, a $(2k - 1)$ -spanner with $O(n^{1+1/k})$ edges can be built in $O(k \cdot n^{2+1/k})$ time.*

Theorem 1.4 [17] [integer-weighted graphs] *For any integer-weighted graph $G = (V, E)$ and any integer $k \geq 1$, a $(2k - 1)$ -spanner with expected $O(k \cdot n^{1+1/k})$ edges can be built in $O(\text{SORT}(m))$ time, where $\text{SORT}(m)$ is the time needed to sort m integers.*

Remark: The algorithm of [17] is randomized: while the guarantees $2k - 1$ and $O(\text{SORT}(m))$ on the stretch and running time, respectively, are deterministic, the guarantee $O(k \cdot n^{1+1/k})$ on the size of the

spanner is in expectation. The fastest known randomized [26] (respectively, deterministic [25]) algorithm for sorting m integers requires expected $O(m \cdot \sqrt{\log \log n})$ (resp., worst-case $O(m \cdot \log \log n)$) running time.

We will henceforth refer to the algorithms of Theorems 1.1, 1.2, 1.3 and 1.4 as Algorithms *UnwtdSp*, *WtdSp*, *WtdSp₂*, and *IntWtdSp*, respectively.

2 The Basic Construction

2.1 The Algorithm

In this section we devise an algorithm *LightSp* that builds a light spanner efficiently.

Let $G = (V, E)$ be an arbitrary weighted graph, with $n = |V|, m = |E|$. Let $k \geq 1$ be an integer parameter that determines the stretch bound of the spanner.

We start with building an MST (or an $O(1)$ -approximate MST) $T = (V, E_T)$ for G . Let $M_T = (V, \text{dist}_T)$ be the (shortest-path) metric induced by T . We then compute the Hamiltonian path $\mathcal{L} = (v_1, v_2, \dots, v_n)$ of M_T drawn by taking the preorder traversal of T . Observe that for each $1 \leq i \leq n - 1$, $\text{dist}_{\mathcal{L}}(v_i, v_{i+1}) = \text{dist}_T(v_i, v_{i+1})$. More generally, for any pair $u, v \in V$ of vertices, $\text{dist}_{\mathcal{L}}(u, v) \geq \text{dist}_T(u, v)$. Define $L = \omega(\mathcal{L})$; it is well known ([14], ch. 36) that $L \leq 2 \cdot \omega(T)$, and so $L = O(\omega(\text{MST}(G)))$.

Let $1 < \rho \leq 2$ be some parameter to be determined later, and define $\ell = \lceil \log_{\rho} n \rceil$. We partition the edges of E into $\ell + 1$ edge sets. The first edge set E_0 contains all edges with weight in the range $W_0 = (0, \frac{L}{n}]$. For each $1 \leq j \leq \ell$, the j th edge set E_j contains all edges with weight in the range $W_j = (\xi_j, \rho \cdot \xi_j]$, where $\xi_j = \rho^{j-1} \cdot \frac{L}{n}$.

Define $V_0 = V$, and let $n_0 = |V_0| = n$. We use Algorithm *WtdSp* to build a $(2k - 1)$ -spanner $H'_0 = (V_0, E'_0)$ for $G_0 = (V, E_0)$.

Algorithm *LightSp* proceeds in ℓ iterations $j = 1, 2, \dots, \ell$.

1. First, we divide the path \mathcal{L} into $n_j = \frac{q \cdot L}{\xi_j} = \frac{q \cdot n}{\rho^{j-1}}$ intervals of length $\mu_j = \frac{\xi_j}{q}$ each, where $\frac{1}{2k-1} < q < k$ is some parameter to be determined later. (Notice that $n_j > n$, for all $1 \leq j < \log_{\rho} q + 1$.) These intervals induce a partition of V in the obvious² way; denote these intervals and the corresponding vertex sets by $I_j^{(1)}, I_j^{(2)}, \dots, I_j^{(n_j)}$ and $V_j^{(1)}, V_j^{(2)}, \dots, V_j^{(n_j)}$, respectively. While computing this partition of V , we will store the index i of the vertex set $V_j^{(i)}$ to which any vertex $v \in V$ belongs in some variable $\text{ind}_j(v)$, for each $1 \leq i \leq n_j$. For each interval $I_j^{(i)}$, $1 \leq i \leq n_j$, we define a new “dummy” vertex $r_j^{(i)}$ (i.e., not present in G) which will serve as the *representative* of the interval $I_j^{(i)}$. The vertices $r_j^{(i)}$ (respectively, intervals $I_j^{(i)}$), $1 \leq i \leq n_j$, will also be referred to as the *j-level representatives* (resp., *j-level intervals*). For any vertex $v \in V$, its *j-level representative* is given by $r_j(v) = r_j^{(\text{ind}_j(v))}$. We also define a dummy vertex set V_j , which contains all the *j-level representatives*. Observe that $|V_j| \leq \min\{n, n_j\} = \min\left\{n, \frac{q \cdot n}{\rho^{j-1}}\right\}$.
2. We then compute an edge set \tilde{E}_j over V_j as follows. First, we remove from the j th edge set E_j all edges that have both their endpoints in the same *j-level interval* to obtain the edge set \bar{E}_j . For each $e = (u, v) \in \bar{E}_j$, let $\hat{e} = (r_j(u), r_j(v))$ be a dummy edge of weight $\omega(e)$ between the corresponding *j-level representatives*, and note that $r_j(u) \neq r_j(v)$; we say that $\hat{e} = r(e)$ is the *representative edge* of e , and that $e = s(\hat{e})$ is the *source edge* of \hat{e} . Define $\hat{E}_j = \{\hat{e} = r(e) \mid e \in \bar{E}_j\}$, and let $\tilde{G}_j = (V_j, \hat{E}_j)$ be the corresponding multi-graph; note that \tilde{G}_j does not contain self-loops. Next, we transform \tilde{G}_j into a simple graph $\hat{G}_j = (V_j, \tilde{E}_j)$ by removing all edges but the one of minimum weight, for every pair of incident vertices in \tilde{G}_j . During this process, we store with every edge $\tilde{e} \in \tilde{E}_j$ its source edge $e = s(\tilde{e}) \in \bar{E}_j$.
3. We proceed to building a $(2k - 1)$ -spanner $H'_j = (V_j, E'_j)$ for $\tilde{G}_j = (V_j, \tilde{E}_j)$ using Algorithm *WtdSp*.

²A vertex that “lies” on the boundary of two consecutive intervals can be assigned to either one of them arbitrarily.

4. Next, we replace each edge $e' \in E'_j$ by its source edge $s(e')$. (Note that $e' \in E'_j \subseteq \tilde{E}_j$ is a dummy edge that connects some two distinct j -level intervals; denote them by I and I' . Moreover, its weight $\omega(e')$ is the smallest weight of an E_j -edge which connects I and I' . Hence the source edge $s(e')$ is the minimum weight E_j -edge which connects the intervals I and I' .) Denote the resulting edge set by E_j^* and define $H_j^* = (V_j, E_j^*)$; note that $E_j^* \subseteq \bar{E}_j \subseteq E_j$. We will refer to H_j^* as the j -level spanner.

Finally, we define $E^* = E_T \cup E'_0 \cup \bigcup_{j=1}^{\ell} E_j^*$, and return the graph $H^* = (V, E^*)$ as our spanner.

2.2 The Analysis

In this section we analyze the properties of the constructed graph H^* .

2.2.1 Stretch. We start with analyzing the stretch of the graph $H^* = (V, E^*)$.

Define $\text{str}(k, q) = (2k - 1) \cdot (1 + \frac{2}{q}) + \frac{2}{q}$. Next, we show that the stretch of H^* is at most $\text{str}(k, q)$.

In other words, we prove that $\text{dist}_{H^*}(u, v) \leq \text{str}(k, q) \cdot \text{dist}_G(u, v)$, for any edge $e = (u, v) \in E$.

If $e \in E_0$, then we have $\text{dist}_{H^*}(u, v) \leq \text{dist}_{H'_0}(u, v) \leq (2k - 1) \cdot \text{dist}_G(u, v) \leq \text{str}(k, q) \cdot \text{dist}_G(u, v)$.

We henceforth assume that $e \in E_j$, for some $1 \leq j \leq \ell$. The next lemma shows that $\text{dist}_{H^*}(u, v) \leq \text{str}(k, q) \cdot \omega(e)$. (This lemma in conjunction with the triangle inequality imply that $\text{dist}_{H^*}(u, v) \leq \text{str}(k, q) \cdot \text{dist}_G(u, v)$, for any edge $e = (u, v) \in E$, which provides the required stretch bound on H^* .)

Lemma 2.1 *There is a path of length at most $\text{str}(k, q) \cdot \omega(e)$ in $H_j^* \cup T = (V, E_j^* \cup E_T)$ between u and v .*

Proof: Since e is in E_j , its weight is in $W_j = [\xi_j, \rho \cdot \xi_j]$. In particular, we have $\omega(e) \geq \xi_j$.

For any pair $x, y \in V$ of vertices, let $\Pi_T(x, y)$ denote the path in the MST T between them.

Suppose first that u and v belong to the same j -level interval. Note that the length of all j -level intervals is $\mu_j = \frac{\xi_j}{q}$, and so $\text{dist}_T(u, v) \leq \text{dist}_{\mathcal{L}}(u, v) \leq \frac{\xi_j}{q}$. Hence, the weight of the path $\Pi_T(u, v)$ between u and v in T is at most

$$\frac{\xi_j}{q} \leq \frac{\omega(e)}{q} < (2k - 1) \cdot \omega(e) < \text{str}(k, q) \cdot \omega(e).$$

(The one before last inequality holds as $q > \frac{1}{2k-1}$.) Thus $\Pi_T(u, v)$ is a path of the required length in $H_j^* \cup T$ between u and v .

We henceforth assume that u and v belong to distinct j -level intervals. Let $\tilde{e} = (r_j(u), r_j(v)) \in \tilde{E}_j$ be the respective edge in \tilde{E}_j , where $r_j(u)$ and $r_j(v)$ are the j -level representatives of u and v , respectively. By the construction of \tilde{E}_j , we have $\omega(\tilde{e}) \leq \omega(e)$. Let $\Pi'(r_j(u), r_j(v)) = (r_j(u) = u'_0, u'_1, \dots, u'_h = r_j(v))$ be a path of length at most $(2k - 1) \cdot \text{dist}_{\tilde{G}_j}(r_j(u), r_j(v))$ in H'_j between $r_j(u)$ and $r_j(v)$. We have

$$\omega(\Pi'(r_j(u), r_j(v))) \leq (2k - 1) \cdot \text{dist}_{\tilde{G}_j}(r_j(u), r_j(v)) \leq (2k - 1) \cdot \omega(\tilde{e}) \leq (2k - 1) \cdot \omega(e). \quad (1)$$

Notice that the edges of H'_j (and of $\Pi'(r_j(u), r_j(v))$) are not taken as is to the graph H_j^* . That is, each edge $e'_i = (u'_i, u'_{i+1}) \in H'_j$ is replaced by its source edge $s(e'_i) = (u_i, u_{i+1}) \in E_j^*$. To connect the two endpoints u'_i and u'_{i+1} of the original edge $e'_i \in E'_j$ in $H_j^* \cup T$, we take the path $\Pi(u'_i, u'_{i+1}) = \Pi_T(u'_i, u_i) \circ (u_i, u_{i+1}) \circ \Pi_T(u_{i+1}, u'_{i+1})$ obtained from the concatenation of the path $\Pi_T(u'_i, u_i)$, the edge $s(e'_i) = (u_i, u_{i+1})$, and the path $\Pi_T(u_{i+1}, u'_{i+1})$. Observe that u'_i is the j -level representative of u_i , i.e., $u'_i = r_j(u_i)$. In particular, they belong to the same j -level interval, and so $\omega(\Pi_T(u'_i, u_i)) \leq \frac{\xi_j}{q}$. Similarly, $\omega(\Pi_T(u_{i+1}, u'_{i+1})) \leq \frac{\xi_j}{q}$. Also, since $s(e'_i) \in E_j^* \subseteq E_j$, it holds that $\omega(e'_i) \geq \xi_j$. Recall that the weight of the source edge $s(e'_i) = (u_i, u_{i+1})$ of e'_i is equal to that of $e'_i = (u'_i, u'_{i+1})$, and so $\omega(s(e'_i)) = \omega(e'_i) \geq \xi_j$. Consequently, the weight $\omega(\Pi(u'_i, u'_{i+1}))$ of the path $\Pi(u'_i, u'_{i+1})$ satisfies

$$\begin{aligned} \omega(\Pi(u'_i, u'_{i+1})) &= \omega(\Pi_T(u'_i, u_i)) + \omega(u_i, u_{i+1}) + \omega(\Pi_T(u_{i+1}, u'_{i+1})) \\ &\leq \frac{\xi_j}{q} + \omega(e'_i) + \frac{\xi_j}{q} \leq \omega(e'_i) \cdot \left(1 + \frac{2}{q}\right). \end{aligned} \quad (2)$$

Let $\Pi^*(r_j(u), r_j(v))$ be the path in $H_j^* \cup T$ between $r_j(u)$ and $r_j(v)$ obtained from $\Pi'(r_j(u), r_j(v))$ by replacing each edge (u'_i, u'_{i+1}) in it by the path $\Pi(u'_i, u'_{i+1})$ as described above. (See Figure 1.(I).) Equations (1) and (2) yield

$$\omega(\Pi^*(r_j(u), r_j(v))) \leq \omega(\Pi'(r_j(u), r_j(v))) \cdot \left(1 + \frac{2}{q}\right) \leq (2k-1) \cdot \omega(e) \cdot \left(1 + \frac{2}{q}\right).$$

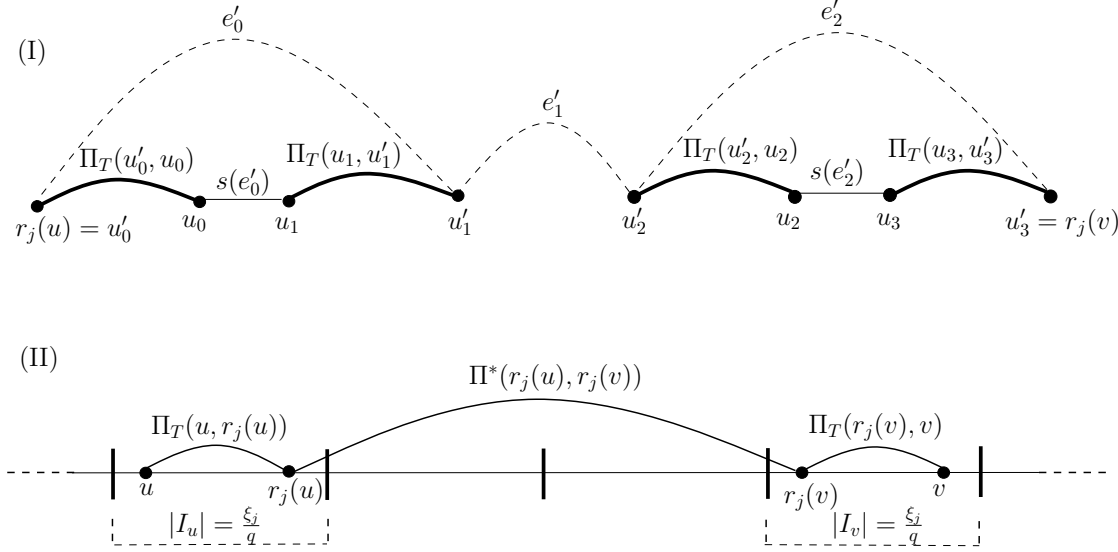


Figure 1: (I) An illustration of the path $\Pi^*(r_j(u), r_j(v))$. Edges $e'_i = (u'_i, u'_{i+1})$ of the path $\Pi'(r_j(u), r_j(v))$ in H_j' are depicted by dashed curved lines. The MST-paths $\Pi_T(u'_i, u_i) \in T$ between vertices u'_i and u_i that belong to the same interval are depicted by thick curved lines. All the other edges (i.e., edges (u_i, u_{i+1}) in H_j^*) are depicted by thin straight lines. (II) An illustration of the path $\Pi^*(u, v)$. The vertices u and $r_j(u)$ (respectively, v and $r_j(v)$) belong to the same j -level interval, denoted here by I_u (resp., I_v); the lengths of these intervals (denoted here by $|I_u|$ and $|I_v|$) are both equal to $\frac{\xi_j}{q}$.

Finally, let $\Pi^*(u, v) = \Pi_T(u, r_j(u)) \circ \Pi^*(r_j(u), r_j(v)) \circ \Pi_T(r_j(v), v)$ be the path in $H_j^* \cup T$ between u and v obtained from the concatenation of the paths $\Pi_T(u, r_j(u))$, $\Pi^*(r_j(u), r_j(v))$ and $\Pi_T(r_j(v), v)$. (See Figure 1.(II) for an illustration.) Since u and $r_j(u)$ belong to the same j -level interval, we have $\omega(\Pi_T(u, r_j(u))) \leq \frac{\xi_j}{q}$. Similarly, $\omega(\Pi_T(r_j(v), v)) \leq \frac{\xi_j}{q}$. Recall that $\omega(e) \geq \xi_j$. We conclude that

$$\begin{aligned} \omega(\Pi^*(u, v)) &= \omega(\Pi_T(u, r_j(u))) + \omega(\Pi^*(r_j(u), r_j(v))) + \omega(\Pi_T(r_j(v), v)) \\ &\leq \frac{\xi_j}{q} + (2k-1) \cdot \omega(e) \cdot \left(1 + \frac{2}{q}\right) + \frac{\xi_j}{q} \leq \omega(e) \cdot \left((2k-1) \cdot \left(1 + \frac{2}{q}\right) + \frac{2}{q}\right) \\ &= \text{str}(k, q) \cdot \omega(e). \end{aligned}$$

Hence $\Pi^*(u, v)$ is a path of length at most $\text{str}(k, q) \cdot \omega(e)$ between u and v in $H_j^* \cup T$, as required. \blacksquare

2.2.2 Number of Edges. The next lemma bounds the number of edges in the spanner $H^* = (V, E^*)$.

Lemma 2.2 $|E^*| = O\left(k \cdot n^{1+1/k} \cdot \left(1 + \frac{q}{\rho-1}\right)\right).$

Proof: Recall that $|V_j| \leq \min\{n, n_j\} = \min\left\{n, \frac{q \cdot n}{\rho^{j-1}}\right\} \leq n$, for $1 \leq j \leq \ell$. For an index $\log_\rho q + 1 \leq j \leq \ell$, we have $\frac{q}{\rho^{j-1}} \leq 1$. Hence $n \geq \frac{q \cdot n}{\rho^{j-1}} = n_j$, and so $|V_j| \leq n_j$. By construction, $E^* = E_T \cup E'_0 \cup \bigcup_{j=1}^\ell E_j^*$. Clearly $|E_T| = n - 1$. By Theorem 1.2, $|E'_0| = O(k \cdot |V_0|^{1+1/k}) = O(k \cdot n^{1+1/k})$.

Also, it holds that $|E_j^*| = |E'_j| = O(k \cdot |V_j|^{1+1/k}) = O(k \cdot n^{1+1/k})$, for each $1 \leq j \leq \ell$. Next, consider an index $\log_\rho q + 1 \leq j \leq \ell$; in this case $|V_j| \leq n_j = \frac{q \cdot n}{\rho^{j-1}}$. Since $q < k$, we have $q^{1/k} = O(1)$. It follows that

$$|E_j^*| = O\left(k \cdot \left(\frac{q \cdot n}{\rho^{j-1}}\right)^{1+1/k}\right) = O\left(k \cdot q \cdot n^{1+1/k} \cdot \left(\frac{1}{\rho^{1+1/k}}\right)^{j-1}\right). \quad (3)$$

Since $\rho > 1$, we have

$$\sum_{\log_\rho q + 1 \leq j \leq \ell} \left(\frac{1}{\rho^{1+1/k}}\right)^{j-1} \leq \sum_{i=0}^{\infty} \left(\frac{1}{\rho^{1+1/k}}\right)^i \leq \sum_{i=0}^{\infty} \left(\frac{1}{\rho}\right)^i = \frac{1}{\rho - 1} + 1.$$

It follows that

$$\begin{aligned} |E^*| &= |E_T| + |E'_0| + \sum_{j=1}^{\ell} |E_j^*| = n - 1 + O(k \cdot n^{1+1/k}) + \sum_{1 \leq j < \log_\rho q + 1} |E_j^*| + \sum_{\log_\rho q + 1 \leq j \leq \ell} |E_j^*| \\ &= O(k \cdot n^{1+1/k}) + O(\log_\rho q \cdot k \cdot n^{1+1/k}) + O\left(k \cdot q \cdot n^{1+1/k}\right) \cdot \sum_{\log_\rho q + 1 \leq j \leq \ell} \left(\frac{1}{\rho^{1+1/k}}\right)^{j-1} \\ &= O\left(k \cdot n^{1+1/k} \cdot \left(1 + \frac{q}{\rho - 1}\right)\right). \end{aligned}$$

(Note that for $q > 1$, it holds that $\log_\rho q = O(\frac{q}{\rho - 1})$.) ■

2.2.3 Weight. The next lemma estimates the weight of $H^* = (V, E^*)$.

Lemma 2.3 $\omega(E^*) = O\left(k^2 \cdot n^{1/k} \cdot \frac{q}{\rho - 1}\right) \cdot \omega(MST(G))$.

Proof: First, observe that the edge weights in E_0 are bounded above by $\frac{L}{n}$. Since $|E'_0| = O(k \cdot n^{1+1/k})$, it follows that

$$\omega(E'_0) \leq O(k \cdot n^{1+1/k}) \cdot \frac{L}{n} = O(k \cdot n^{1/k} \cdot L). \quad (4)$$

The edge weights in E_j are bounded above by $\xi_j \cdot \rho = \rho^j \cdot \frac{L}{n}$, for each $1 \leq j \leq \ell$. Consider first indices j in the range $1 \leq j < \log_\rho q + 1$. Since $|E_j^*| = O(k \cdot n^{1+1/k})$, we have $\omega(E_j^*) = O(k \cdot n^{1+1/k}) \cdot (\rho^j \cdot \frac{L}{n}) = O(k \cdot n^{1/k} \cdot \rho^j \cdot L)$. It follows that

$$\sum_{1 \leq j < \log_\rho q + 1} \omega(E_j^*) = O\left(k \cdot n^{1/k} \cdot L\right) \cdot \sum_{1 \leq j < \log_\rho q + 1} \rho^j = O\left(k \cdot n^{1/k} \cdot L \cdot q\right). \quad (5)$$

Next, consider indices j with $\log_\rho q + 1 \leq j \leq \ell$. By (3), $|E_j^*| = O\left(k \cdot q \cdot n^{1+1/k} \cdot \left(\frac{1}{\rho^{1+1/k}}\right)^{j-1}\right)$. Hence

$$\omega(E_j^*) = O\left(k \cdot q \cdot n^{1+1/k} \cdot \left(\frac{1}{\rho^{1+1/k}}\right)^{j-1}\right) \cdot \left(\rho^j \cdot \frac{L}{n}\right) = O\left(k \cdot q \cdot n^{1/k} \cdot \left(\frac{1}{\rho^{1/k}}\right)^{j-1} \cdot L\right).$$

Observe that

$$\sum_{\log_\rho q + 1 \leq j \leq \ell} \left(\frac{1}{\rho^{1/k}}\right)^{j-1} \leq \sum_{i=0}^{\infty} \left(\frac{1}{\rho^{1/k}}\right)^i = O\left(\frac{k}{\rho - 1}\right).$$

It follows that

$$\sum_{\log_\rho q + 1 \leq j \leq \ell} \omega(E_j^*) = O\left(k \cdot q \cdot n^{1/k} \cdot L\right) \cdot \sum_{\log_\rho q + 1 \leq j \leq \ell} \left(\frac{1}{\rho^{1/k}}\right)^{j-1} = O\left(k^2 \cdot q \cdot n^{1/k} \cdot L \cdot \frac{1}{\rho - 1}\right). \quad (6)$$

By Equations (4), (5) and (6),

$$\begin{aligned}
\omega(E^*) &= \omega(T) + \omega(E'_0) + \sum_{1 \leq j < \log_\rho q + 1} \omega(E_j^*) + \sum_{\log_\rho q + 1 \leq j \leq \ell} \omega(E_j^*) \\
&= \omega(T) + O(k \cdot n^{1/k} \cdot L) + O\left(k \cdot n^{1/k} \cdot L \cdot q\right) + O\left(k^2 \cdot q \cdot n^{1/k} \cdot L \cdot \frac{1}{\rho - 1}\right) \\
&= O\left(k^2 \cdot n^{1/k} \cdot \frac{q}{\rho - 1}\right) \cdot \omega(MST(G)).
\end{aligned}$$

(Since $q > \frac{1}{2k-1}$, we have $k^2 \cdot q = \Omega(k)$.) ■

2.2.4 Running Time. The next lemma bounds the running time of our construction.

Lemma 2.4 *The graph H^* can be built within time $O(k \cdot m + \min\{n \cdot \log n, m \cdot \alpha(n)\})$.*

Proof: The tree T can be built in time $O(\min\{m + n \cdot \log n, m \cdot \alpha(m, n)\})$ [12]. Computing the Hamiltonian path \mathcal{L} of M_T , as well as its weight $L = \omega(\mathcal{L})$, take another $O(n)$ time. Having computed L , we can partition the edges of E to the $\ell + 1$ edge sets E_0, E_1, \dots, E_ℓ in $O(m)$ time in the obvious way. By Theorem 1.2, building the spanner H'_0 for $G_0 = (V, E_0)$ requires $O(k \cdot |E_0|) = O(k \cdot m)$ time.

Next, we bound the running time of a single iteration j in the main loop, for $1 \leq j \leq \ell$.

1. For each $1 \leq j \leq \ell$, we allocate an array A_j of size n_j . For each $x \in V_j$, the entry $A_j[x]$ will contain the linked list of x 's neighbors in the multi-graph \hat{G}_j (with the weights of the representative edges). For each edge $e = (u, v) \in E$ we determine the index $j \in [\ell]$ such that $\omega(e) \in W_j$ (and so $e \in E_j$). By the locations of u and v on the path \mathcal{L} , we determine the j -level representatives $r_j(u)$ and $r_j(v)$ of u and v , respectively. We also determine whether the edge crosses between different j -level intervals. If it is the case (i.e., $e \in \tilde{E}_j$), then we insert the edge into the array A_j . (In other words, we update the linked lists of both $r_j(u)$ and $r_j(v)$ in A_j .) In this way we form the multi-graphs $\hat{G}_1, \hat{G}_2, \dots, \hat{G}_\ell$ within $O(|E|) = O(m)$ time. To prune these multi-graphs into simple graphs $\tilde{G}_1, \tilde{G}_2, \dots, \tilde{G}_\ell$, we need to remove all edges but the one of minimum weight, for every pair of incident vertices in \hat{G}_j . This can be done deterministically within $O(m)$ time and space. (Alternatively, one can apply one of the algorithms for constructing $(2k - 1)$ -spanners with $O(k \cdot n^{1+1/k})$ edges for weighted graphs [9, 34, 17] directly on the multi-graphs $\hat{G}_1, \hat{G}_2, \dots, \hat{G}_\ell$, without pruning them first. It is easy to verify that this does not affect their guarantees for stretch, size and running time.)
2. By Theorem 1.2, the time needed to build the $(2k - 1)$ -spanner $H'_j = (V_j, E'_j)$ for $\tilde{G}_j = (V, \tilde{E}_j)$ is $O(k \cdot |\tilde{E}_j|) = O(k \cdot |E_j|)$. Summing over all ℓ iterations, the overall time is $\sum_{j=1}^{\ell} O(k \cdot |E_j|) = O(k \cdot m)$.
3. Replacing each edge of E'_j by its source edge takes $O(1)$ time, thus the graph $H_j^* = (V, E_j^*)$ is built in $O(|E'_j|) = O(|E_j|)$ time. Summing over all ℓ iterations, the overall time is $\sum_{j=1}^{\ell} O(|E_j|) = O(m)$.

It follows that the total running time of the construction is

$$O(\min\{m + n \cdot \log n, m \cdot \alpha(m, n)\}) + O(k \cdot m) = O(k \cdot m + \min\{n \cdot \log n, m \cdot \alpha(n)\}). \quad \blacksquare$$

We remark that there are randomized algorithms for constructing MST within $O(m + n)$ time [27]. Employing one of them reduces the running time of our algorithm to just $O(k \cdot m)$.

2.2.5 Summary. We summarize the properties of the spanner $H^* = (V, E^*)$ in the following theorem. (We substituted $\rho = 2$ to optimize the parameters of the construction.)

Theorem 2.5 *Let $G = (V, E)$ be a weighted graph, with $n = |V|, m = |E|$. For any integer $k \geq 1$ and any number $\frac{1}{2k-1} < q < k$, a $\left((2k - 1) \cdot (1 + \frac{2}{q}) + \frac{2}{q}\right)$ -spanner with $O(k \cdot n^{1+1/k} \cdot (1 + q))$ edges and lightness $O(k^2 \cdot n^{1/k} \cdot q)$, can be built in $O(k \cdot m + \min\{n \cdot \log n, m \cdot \alpha(n)\})$ time.*

By substituting $q = \Theta(\frac{1}{\epsilon})$ in Theorem 2.5, for some small constant $\epsilon > 0$, we obtain:

Corollary 2.6 *Let $G = (V, E)$ be a weighted graph, with $n = |V|, m = |E|$. For any integer $k \geq 1$ and any constant $\epsilon > 0$, a $((2k - 1) \cdot (1 + \epsilon))$ -spanner with $O(k \cdot n^{1+1/k})$ edges and lightness $O(k^2 \cdot n^{1/k})$, can be built in $O(k \cdot m + \min\{n \cdot \log n, m \cdot \alpha(n)\})$ time.*

Note also that by substituting $q = \Theta(k/\epsilon)$, we get a $(2k - 1 + \epsilon)$ -spanner with $O(k^2 \cdot n^{1+1/k})$ edges and lightness $O(k^3 \cdot n^{1/k})$, within the same time.

3 Variants of the Construction

In this section we devise some variants of the basic construction of Section 2 by applying a small (but significant) modification to Algorithm *LightSp*.

Notice that Algorithm *LightSp* employs Algorithm *WtdSp* as a black box for building (i) the spanner $H'_0 = (V_0, E'_0)$ for the graph $G_0 = (V, E_0)$, and (ii) the spanner $H'_j = (V_j, E'_j)$ for the graph $\tilde{G}_j = (V_j, \tilde{E}_j)$ in step 3 of the main loop, for each $1 \leq j \leq \ell$. Instead of employing Algorithm *WtdSp*, we can employ some of the other black-box spanners that are summarized in Section 1.6.

3.1 First Variant (Slightly Increasing the Stretch)

There is a significant difference between the spanner H'_0 , and the spanners H'_j for $j \geq 1$. While the aspect ratio (defined as the ratio between the maximum and minimum edge weights) of the graph G_0 may be unbounded, the aspect ratio of all the graphs $\tilde{G}_j, 1 \leq j \leq \ell$, is bounded above by ρ . This suggests that we may view the graphs \tilde{G}_j as unweighted, and will thus be able to use Algorithm *UnwtdSp* for building the spanners H'_j for them. As a result, the stretch of each spanner H'_j will increase by a factor of ρ .

Denote by \mathcal{H}^* the variant of \mathcal{H} obtained by employing Algorithm *UnwtdSp* for building the spanners $H'_j, 1 \leq j \leq \ell$. (We still use Algorithm *WtdSp* to build the spanner H'_0 .)

We will next analyze the properties of the resulting construction \mathcal{H}^* . This analysis is very similar to the analysis of the basic construction that is given in Section 2.2, hence we aim for conciseness.

Stretch. We first show how to adapt the stretch analysis of Section 2.2.1 to the new construction \mathcal{H}^* . The proof of Lemma 2.1 carries through except for Equation (1) that needs to be changed. Observe that all weights of edges in \tilde{E}_j belong to $W_j = [\xi_j, \rho \cdot \xi_j]$. Hence they differ from each other by at most a multiplicative factor of ρ . Note that H'_j is an *unweighted* $(2k - 1)$ -spanner for \tilde{G}_j , i.e., a subgraph of \tilde{G}_j that contains, for every edge $e \in \tilde{E}_j$, a path with at most $(2k - 1)$ edges between its endpoints. It is easy to see that H'_j provides a $(\rho \cdot (2k - 1))$ -spanner for \tilde{G}_j . Hence, instead of Equation (1), we will now have

$$\omega(\Pi'(r_j(u), r_j(v))) \leq \rho \cdot (2k - 1) \cdot \text{dist}_{\tilde{G}_j}(r_j(u), r_j(v)) \leq \rho \cdot (2k - 1) \cdot \omega(\tilde{e}) \leq \rho \cdot (2k - 1) \cdot \omega(e).$$

As a result, the upper bound on the weight of the path $\Pi^*(r_j(u), r_j(v))$ will also increase by a factor of ρ , and we will have $\omega(\Pi^*(r_j(u), r_j(v))) \leq \rho \cdot (2k - 1) \cdot \omega(e) \cdot \left(1 + \frac{2}{q}\right)$. Consequently, we will get that $\omega(\Pi^*(u, v)) \leq \omega(e) \cdot \left(\rho \cdot (2k - 1) \cdot \left(1 + \frac{2}{q}\right) + \frac{2}{q}\right)$. Hence the stretch of \mathcal{H}^* is $\rho \cdot (2k - 1) \cdot \left(1 + \frac{2}{q}\right) + \frac{2}{q}$.

Number of Edges. Next, we show how to adapt the size analysis for the new construction \mathcal{H}^* .

The only change from the proof of Lemma 2.2 is that now the upper bound on $|E_j^*| = |E'_j|$, for $1 \leq j \leq \ell$, is smaller by a factor of k . The reason is that we now use Algorithm *UnwtdSp* rather than Algorithm *WtdSp* to build the spanner $H'_j = (V_j, E'_j)$, for $1 \leq j \leq \ell$; compare Theorem 1.1 with Theorem 1.2. Note that the bound on $|E'_0|$ remains unchanged, though, as we still use Algorithm *WtdSp* to build the spanner H'_0 . We will get

$$|E^*| = |E_T| + |E'_0| + \sum_{j=1}^{\ell} |E_j^*| = n - 1 + O(k \cdot n^{1+1/k}) + \sum_{1 \leq j < \log_{\rho} q + 1} |E_j^*| + \sum_{\log_{\rho} q + 1 \leq j \leq \ell} |E_j^*|$$

$$\begin{aligned}
&= O(k \cdot n^{1+1/k}) + O(\log_\rho q \cdot n^{1+1/k}) + O\left(q \cdot n^{1+1/k}\right) \cdot \sum_{\log_\rho q+1 \leq j \leq \ell} \left(\frac{1}{\rho^{1+1/k}}\right)^{j-1} \\
&= O\left(n^{1+1/k} \cdot \left(k + \frac{q}{\rho-1}\right)\right).
\end{aligned}$$

Weight. We now show how to adapt the weight analysis for the new construction \mathcal{H}^* .

The only change from the proof of Lemma 2.3 is that now the upper bound on $\omega(E_j^*)$, $1 \leq j \leq \ell$, is smaller by a factor of k . The bound on $\omega(E_0')$ remains unchanged (see Equation (4)). Hence,

$$\begin{aligned}
\omega(E^*) &= \omega(T) + \omega(E_0') + \sum_{1 \leq j < \log_\rho q+1} \omega(E_j^*) + \sum_{\log_\rho q+1 \leq j \leq \ell} \omega(E_j^*) \\
&= \omega(T) + O(k \cdot n^{1/k} \cdot L) + O\left(n^{1/k} \cdot L \cdot q\right) + O\left(k \cdot q \cdot n^{1/k} \cdot L \cdot \frac{1}{\rho-1}\right) \\
&= O\left(k \cdot n^{1/k} \cdot \left(1 + \frac{q}{\rho-1}\right)\right) \cdot \omega(MST(G)).
\end{aligned}$$

Running time. Clearly, the running time of \mathcal{H}^* is not higher than that of the basic construction.

We summarize the properties of the resulting construction \mathcal{H}^* in the following theorem.

Theorem 3.1 *Let $G = (V, E)$ be a weighted graph, with $n = |V|, m = |E|$. For any integer $k \geq 1$, and any numbers $\frac{1}{2k-1} < q < k$ and $1 < \rho \leq 2$, a $\left(\rho \cdot (2k-1) \cdot \left(1 + \frac{2}{q}\right) + \frac{2}{q}\right)$ -spanner with $O\left(n^{1+1/k} \cdot \left(k + \frac{q}{\rho-1}\right)\right)$ edges and lightness $O\left(k \cdot n^{1/k} \cdot \left(1 + \frac{q}{\rho-1}\right)\right)$, can be built in $O(k \cdot m + \min\{n \cdot \log n, m \cdot \alpha(n)\})$ time.*

By substituting $q = \Theta(\frac{1}{\epsilon})$, $\rho = 1 + \Theta(\epsilon)$ in Theorem 3.1, for an arbitrary constant $\epsilon > 0$, we obtain:

Corollary 3.2 *Let $G = (V, E)$ be a weighted graph, with $n = |V|, m = |E|$. For any integer $k \geq 1$ and any small constant $\epsilon > 0$, a $((2k-1) \cdot (1 + \epsilon))$ -spanner with $O(k \cdot n^{1+1/k})$ edges and lightness $O(k \cdot n^{1/k})$, can be built in $O(k \cdot m + \min\{n \cdot \log n, m \cdot \alpha(n)\})$ time.*

Corollary 3.2 in the particular case $k = O(\log n)$ gives rise to an $O(\log n)$ -spanner with $O(n \cdot \log n)$ edges and lightness $O(\log n)$. The running time of this construction is $O(m \cdot \log n)$.

3.2 Second Variant (Increasing the Running Time)

Denote by $\tilde{\mathcal{H}}$ the variant of H^* obtained by employing Algorithm *UnwtdSp* for building the spanners H_j' , $1 \leq j \leq \ell$, and employing Algorithm *WtdSp₂* (due to Roditty and Zwick [34]) to build the spanner H_0' . It is easy to see that the stretch bound of the resulting construction $\tilde{\mathcal{H}}$ is equal to that of \mathcal{H}^* . Also, the size and weight bounds of $\tilde{\mathcal{H}}$ are better than those of H^* and \mathcal{H}^* , but the running time (which is dominated by the running time of Algorithm *WtdSp₂*) is higher. The analysis of this variant is very similar to the analysis of the basic construction (in Section 2.2) and the analysis of the first variant (in Section 3.1), and is thus omitted.

We summarize the properties of the resulting construction $\tilde{\mathcal{H}}$ in the following theorem.

Theorem 3.3 *Let $G = (V, E)$ be a weighted graph, with $n = |V|$. For any integer $k \geq 1$, and any numbers $\frac{1}{2k-1} < q < k$ and $1 < \rho \leq 2$, a $\left(\rho \cdot (2k-1) \cdot \left(1 + \frac{2}{q}\right) + \frac{2}{q}\right)$ -spanner with $O\left(n^{1+1/k} \cdot \left(1 + \frac{q}{\rho-1}\right)\right)$ edges and lightness $O\left(k \cdot n^{1/k} \cdot \frac{q}{\rho-1}\right)$, can be built in $O(k \cdot n^{2+1/k})$ time.*

By substituting $q = \Theta(\frac{1}{\epsilon})$, $\rho = 1 + \Theta(\epsilon)$ in Theorem 3.3, for an arbitrary constant $\epsilon > 0$, we obtain:

Corollary 3.4 *Let $G = (V, E)$ be a weighted graph, with $n = |V|$. For any integer $k \geq 1$ and any small constant $\epsilon > 0$, a $((2k-1) \cdot (1 + \epsilon))$ -spanner with $O(n^{1+1/k})$ edges and lightness $O(k \cdot n^{1/k})$, can be built in $O(k \cdot n^{2+1/k})$ time.*

Note that the tradeoff between the stretch and lightness exhibited in Corollary 3.4 is the same as in [11]. (There the stretch is $2k - 1$, and the lightness is $O(k \cdot n^{(1+\epsilon)/k})$.) The number of edges is $O(n^{1+1/k})$ in both results, but our running time is $O(k \cdot n^{2+1/k})$, instead of $O(m \cdot (n^{1+1/k} + n \cdot \log n))$ in [11].

By substituting $q = \Theta(1/k)$, $\rho = 2$ in Theorem 3.3, we obtain:

Corollary 3.5 *Let $G = (V, E)$ be a weighted graph, with $n = |V|$. For any integer $k \geq 1$, an $O(k^2)$ -spanner with $O(n^{1+1/k})$ edges and lightness $O(n^{1/k})$, can be built in $O(k \cdot n^{2+1/k})$ time.*

Corollary 3.5 in the particular case $k = O(\log n)$ gives rise to an $O(\log^2 n)$ -spanner with $O(n)$ edges and lightness $O(1)$. We can extend this result to get a general tradeoff between the three parameters by substituting $k = O(\log n)$, $\rho = 2$ in Theorem 3.3, and taking $1 \leq \ell = \frac{1}{q} = O(\log n)$.

Corollary 3.6 *Let $G = (V, E)$ be a weighted graph, with $n = |V|$. For any number $1 \leq \ell = O(\log n)$, an $O(\log n \cdot \ell)$ -spanner with $O(n)$ edges and lightness $O(\frac{\log n}{\ell})$, can be built in $O(n^2 \cdot \log n)$ time.*

The tradeoff of Corollary 3.6 was also given by Chandra et al. [11], but their running time is $O(m \cdot n \cdot \log n)$.

3.3 Third Variant (Increasing the Running Time Some More)

Denote by $\hat{\mathcal{H}}$ the variant of H^* obtained by employing Algorithm *WtdSp₂* for building all the spanners H'_j , $1 \leq j \leq \ell$, as well as the spanner H'_0 . It is easy to see that the stretch bound of the resulting construction $\hat{\mathcal{H}}$ is equal to that of the basic construction H^* . Also, the size and weight bounds of $\hat{\mathcal{H}}$ are exactly the same as those of the second variant $\tilde{\mathcal{H}}$, but the running time is slightly higher (by a factor of $(1 + q^2)$) than that of $\tilde{\mathcal{H}}$. The analysis of this variant is very similar to the above, and is thus omitted.

We summarize the properties of the resulting construction $\hat{\mathcal{H}}$ in the following theorem. (We substituted $\rho = 2$ to optimize the parameters of the construction.)

Theorem 3.7 *Let $G = (V, E)$ be a weighted graph, with $n = |V|$. For any integer $k \geq 1$, and any number $\frac{1}{2k-1} < q < k$, a $\left((2k-1) \cdot (1 + \frac{2}{q}) + \frac{2}{q}\right)$ -spanner with $O(n^{1+1/k} \cdot (1+q))$ edges and lightness $O(k \cdot n^{1/k} \cdot q)$, can be built in $O(k \cdot n^{2+1/k} \cdot (1+q^2))$ time.*

3.4 Fourth Variant (Integer-Weighted Graphs)

The fourth variant of our construction applies to integer-weighted graphs only. Denote by \mathcal{H}_{int} the variant of H^* obtained by employing Algorithm *IntWtdSp* for building all the black-box spanners, i.e., the spanners H'_j , $1 \leq j \leq \ell$, and the spanner H'_0 . The new construction achieves exactly the same bounds as the basic construction H^* , except for the running time. The running time of this variant consists of two parts. Specifically, it is the time required to compute the MST and the time required to compute a spanner. On an integer-weighted graph the first task can be done in $O(m + n)$ time [22], while the second task requires $O(\text{SORT}(m))$ time (see Theorem 1.4 in [17]). Note, however, that the construction becomes randomized.

We summarize the properties of the new construction \mathcal{H}_{int} in the following theorem.

Theorem 3.8 *Let $G = (V, E)$ be an integer-weighted graph, with $n = |V|$, $m = |E|$. For any integer $k \geq 1$ and any number $\frac{1}{2k-1} < q < k$, a $\left((2k-1) \cdot (1 + \frac{2}{q}) + \frac{2}{q}\right)$ -spanner with expected $O(k \cdot n^{1+1/k} \cdot (1+q))$ edges and expected lightness $O(k^2 \cdot n^{1/k} \cdot q)$, can be built in $O(\text{SORT}(m))$ time.*

3.5 Fifth Variant (Spanners in the Streaming Model)

In this section we analyze our algorithm in the augmented streaming model. Specifically, this is the model introduced by Aggarwal et al. [1], which allows sorting passes over the input.

Our algorithm relies on a streaming algorithm for constructing sparse (but possibly heavy) spanners from [17], summarized in Theorem 3.9 below. We remark that Theorem 3.9 also applies to multi-graphs.

Theorem 3.9 [17] *For any unweighted n -vertex graph $G = (V, E)$ and any integer $k \geq 1$, there exists a one-pass streaming algorithm that computes a $(2k - 1)$ -spanner with $O(k \cdot n^{1+1/k})$ edges (expected). The processing time-per-edge of the algorithm is $O(1)$, and its space requirement is $O(k \cdot n^{1+1/k})$ (expected). Also, in the augmented streaming model (if the algorithm accepts a sorted stream of edges as input) the algorithm produces $(2k - 1)$ -spanners with $O(k \cdot n^{1+1/k})$ edges (expected) for weighted graphs as well. The processing time-per-edge and space requirements of this algorithm are the same as in the unweighted case.*

Our algorithm will run $\ell + 1$ copies of the algorithm for weighted graphs from Theorem 3.9. We will denote these copies \mathcal{A}_j , for $0 \leq j \leq \ell$.

Our algorithm starts with a sorting pass over the stream of edges. After this pass, in the consecutive pass the algorithm reads edges in a non-decreasing order of weights. The objective of the first pass (after the sorting pass) is to compute an MST of the input graph. To accomplish this, the algorithm maintains a Union-Find data structure (see Ch. 21 in [14]). It is known [10, 3] that all operations can be performed in worst-case $O(\frac{\log n}{\log \log n})$ time, and in total $O(n + m \cdot \alpha(n))$ time, using $O(n)$ space.

As a result of the first pass, the MST T is computed. The Hamiltonian path \mathcal{L} of M_T is computed between the passes. In addition, the algorithm maintains the location on \mathcal{L} of every vertex $v \in V$. It also initializes $\ell + 1$ arrays A_j of size n_j each, $0 \leq j \leq \ell$.

Then the algorithm performs the second pass over the sorted stream of edges. For each edge $e = (u, v)$ that the algorithm reads in the second pass, the algorithm determines the index j such that $\omega(e) \in W_j$. Then it tests if the edge crosses between different j -level intervals, i.e., belongs to \bar{E}_j . If it does not, this edge is skipped. Otherwise the algorithm passes the edge e to the j th copy \mathcal{A}_j of the weighted streaming algorithm from Theorem 3.9. The streaming algorithm \mathcal{A}_j will ultimately produce the spanner H'_j for $\hat{G}_j = (V_j, \hat{E}_j)$. (Here we follow the notation of Section 2.1.) If the streaming algorithm \mathcal{A}_j decides to insert e into the spanner H'_j , it will also insert its source edge $s(e)$ into the ultimate spanner $H^* = (V, E^*)$. As a result the spanner H^* will contain the union of the j -level spanners, for $0 \leq j \leq \ell$. The edge set of the MST will also be inserted into H^* (either between the passes, or after the second pass).

This completes the description of the algorithm. By Theorem 3.9, the (expected) space requirement in the second pass is $\sum_{j=0}^{\ell} O(k \cdot n_j^{1+1/k}) = O(k \cdot n^{1+1/k})$. The processing time-per-edge is $O(1)$.

We summarize our streaming algorithm in the following theorem.

Theorem 3.10 *In the augmented streaming model, for any weighted graph $G = (V, E)$, any integer $k \geq 1$, and any constant $\epsilon > 0$, our algorithm requires two passes after the sorting pass. It computes a $((2k - 1) \cdot (1 + \epsilon))$ -spanner with expected $O(k \cdot n^{1+1/k})$ edges and expected lightness $O(k^2 \cdot n^{1/k})$. The expected space requirement is $O(k \cdot n^{1+1/k})$. The worst-case processing time-per-edge of the first (respectively, second) pass is $O(\frac{\log n}{\log \log n})$ (resp., $O(1)$). Moreover, the overall processing time of the first pass is $O(m \cdot \alpha(n))$.*

As was mentioned in the introduction, the algorithm of [11] can be viewed as an algorithm in this model. It constructs $(2k - 1)$ -spanners with $O(n^{1+1/k})$ edges and lightness $O(k \cdot n^{(1+\epsilon)/k})$. It requires one pass after the initial sorting pass, but its processing time-per-edge is very large (specifically, it is $O(n^{1+1/k})$).

References

- [1] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *Proc. of 45th FOCS*, pages 540–549, 2004.
- [2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.
- [3] S. Alstrup, A. M. Ben-Amram, and T. Rauhe. Worst-case and amortised optimality in union-find (extended abstract). In *Proc. of 31st STOC*, pages 499–506, 1999.
- [4] I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.
- [5] B. Awerbuch. Communication-time trade-offs in network synchronization. In *Proc. of 4th PODC*, pages 272–276, 1985.

- [6] B. Awerbuch, A. Baratz, and D. Peleg. Efficient broadcast and light-weight spanners. *Technical Report CS92-22, Weizmann Institute*, October, 1992.
- [7] S. Baswana. Streaming algorithm for graph spanners - single pass and constant processing time per edge. *Inf. Process. Lett.*, 106(3):110–114, 2008.
- [8] S. Baswana, T. Kavitha, K. Mehlhorn, and S. Pettie. Additive spanners and (α, β) -spanners. *ACM Transactions on Algorithms*, 7(1):5, 2010.
- [9] S. Baswana and S. Sen. A simple linear time algorithm for computing a $(2k - 1)$ -spanner of $O(n^{1+1/k})$ size in weighted graphs. In *Proc. of 30th ICALP*, pages 384–296, 2003.
- [10] N. Blum. On the single-operation worst-case time complexity of the disjoint set union problem. *SIAM J. Comput.*, 15(4):1021–1024, 1986.
- [11] B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. In *Proc. of 8th SOCG*, pages 192–201, 1992.
- [12] B. Chazelle. The complexity of computing partial sums off-line. *J. ACM*, 47(6):1028–1047, 2000.
- [13] E. Cohen. Fast algorithms for constructing t -spanners and paths with stretch t . In *Proc. of 34th FOCS*, pages 648–658, 1993.
- [14] T. H. Corman, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd edition*. McGraw-Hill Book Company, Boston, MA, 2001.
- [15] G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *Int. J. Comput. Geometry Appl.*, 7(4):297–315, 1997.
- [16] M. Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1(2):283–323, 2005.
- [17] M. Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Transactions on Algorithms*, 7(2):20, 2011.
- [18] M. Elkin and D. Peleg. $(1+\epsilon, \beta)$ -spanner constructions for general graphs. *SIAM J. Comput.*, 33(3):608–631, 2004.
- [19] M. Elkin and J. Zhang. Efficient algorithms for constructing $(1+\epsilon, \beta)$ -spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006.
- [20] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the streaming model: the value of space. In *Proc. of 16th SODA*, pages 745–754, 2005.
- [21] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate l_1 -difference algorithm for massive data streams. *SIAM J. Comput.*, 32(1):131–151, 2002.
- [22] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.
- [23] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM J. Comput.*, 31(5):1479–1500, 2002.
- [24] S. Halperin and U. Zwick. Linear time deterministic algorithm for computing spanners for unweighted graphs. manuscript, 1996.
- [25] Y. Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, 2004.
- [26] Y. Han and M. Thorup. Integer sorting in $O(n\sqrt{\log \log n})$ expected time and linear space. In *Proc. of 43rd FOCS*, pages 135–144, 2002.
- [27] D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, 1995.
- [28] D. Peleg. Proximity-preserving labeling schemes and their applications. In *Proc. of 25th WG*, pages 30–41, 1999.
- [29] D. Peleg and A. Schäffer. Graph spanners. *J. Graph Theory*, 13(1):99–116, 1989.
- [30] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989.
- [31] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.
- [32] S. Pettie. Low distortion spanners. *ACM Transactions on Algorithms*, 6(1), 2009.
- [33] L. Roditty, M. Thorup, and U. Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Proc. of 32nd ICALP*, pages 261–272, 2005.
- [34] L. Roditty and U. Zwick. On dynamic shortest paths problems. In *Proc. of 32nd ESA*, pages 580–591, 2004.
- [35] M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. of 33rd STOC*, pages 183–192, 2001.
- [36] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. of 13th SPAA*, pages 1–10, 2001.
- [37] M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proc. of 17th SODA*, pages 802–809, 2006.
- [38] D. P. Woodruff. Lower bounds for additive spanners, emulators, and more. In *Proc. of 47th FOCS*, pages 389–398, 2006.